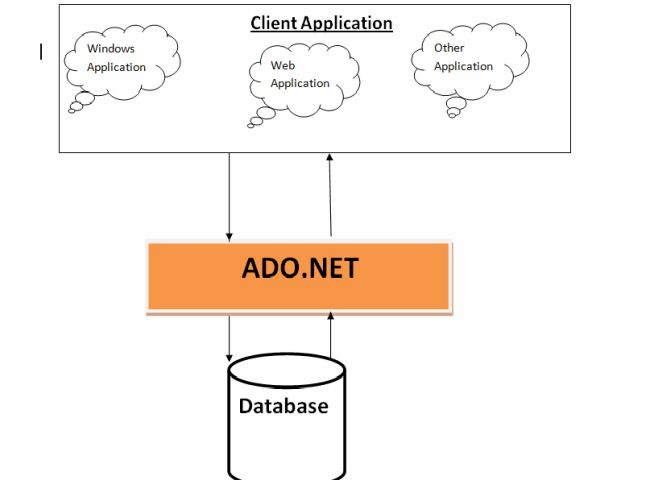
**1.ADO.NET**

ADO stands for **ActiveX Data Object**. ADO.NET is a large set of .NET classes that **enable us** to retrieve, manipulate and update data sources in many ways. As an integral part of the .NET framework, it shares many of its **features** such as multi-language support, garbage collection, just-in-time compilation, object-oriented design, and dynamic caching. **ADO.NET is a part** of the .NET framework architecture.

**ADO.NET Object Model:** It is a model used by .NET applications to communicate with a database for retrieving, accessing, and updating data, as shown in the following figure:



In the ADO.NET object model, the data residing in a database is retrieved through a data provider. The data provider is the set of components including the Connection, Command, DataReader and DataAdapter objects. An application can access data either through a dataset or through a datareader object.

The ADO.NET object model consists of two fundamental components:

* **Data Provider**
* **DataSet**

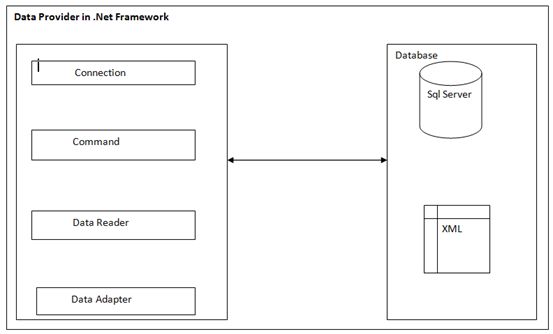
**A. Data Provider:**

Selecting an appropriate data provider for a client application depends on the type of data source being accessed. There are four .Net data providers available.

1. ***SQL Server:*** It is used to work specifically with Microsoft SQL Server. It exists in a namespace within the ***System.Data.SqClient***.
2. ***OLE DB:***OLEDB stands for Object Linking and Embedding Database. It's used to work with the OLEDB provider. The System.Data.dll assembly implements the OLEDB .NET framework data provider in the ***System.Data.OleDb*** namespace.
3. ***ODBC:***ODBC stands for Open Database connectivity. To use this type of provider, you must use an ODBC driver. The ***System.Data.ODBC***.dll assembly implements the ODBC .NET framework data.
4. ***Oracle:*** The System.Data.OracleClient.dll assembly implements the Oracle .NET framework data provider in the ***System.Data.OracleClient*** namespace.

***Data Provider Components****:*The four key components of a dataprovider are:

1. **Connection:** Used to connect to the data source.
2. **Command:** Used to execute a command against the data source. This component retrieves, inserts, deletes and modifies data in a data source.
3. **DataReader:** This component retrieves data from a data source in read-only and forward mode.
4. **DataAdapter:** Used to populate a dataset with the data retrieved from a database and to update the data source.



**B. DataSet:**

The ADO.NET DataSet contains DataTableCollection and their DataRelationCollection. We can use Dataset in combination withDataAdapter Class. The DataSet object offers disconnected data source architecture. The Dataset can work with the data without knowing the source of the data coming from. That is, the Dataset can work with a disconnected mode from its Data Source. It gives a better advantage over DataReader , because the DataReader is working only with the connection oriented Data Sources.

The DataAdapter Object allows us to populate DataTables in a DataSet. We can use Fill method of the DataAdapter for populating data in a Dataset. The DataSet can be filled either from a data source or dynamically.

**2.Importing the SqlClient Namespace**

First, we need to decide what data provider we want to use in our application. Once we have decided, we need to include the namespace in your applications so the namespace objects are available to you. We will include a namespace by adding an Imports directive, followed by the namespace, to your project. The following lines import the System.Data, System.Data.Common, System.Data.SqlClientand System.Data.OleDb namespaces of ADO.NET:

Using System.Data

Using System.Data.Common

Using System.Data.SqlClient

Using System.Data.OleDb

You can also access a namespace's objects by specifying the full namespace when you declare the objects. For example, the following code accesses the SqlConnection class of the System.Data.SqlClient namespace:

***For Example:***

using System;

**using System.Data;**

**using System.Data.SqlClient;**

public partial class Readform : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

SqlConnection = new SqlClient.SqlConnection()

SqlCommand cmd = new SqlCommand("Select \* from Table1", con);

con.Open();

SqlDataReader dr = cmd.ExecuteReader();

con.Close();

}

}

**3.Defining Database Connections (Creating and Managing Connections In ADO.NET)**

**Creating a Connection object**

The connection component of a dataprovider establishes a connection with a data base. To connect to a Microsoft SQL Server, you use the SQL connection class. The following are the commonly used properties and Methods of the SqlConnection class.

**ConnectionString:**provides information, such as database name and, user credentials for database access and so on.

**Open():**  Opens the connection for accessing the database.

**Close():** Closes the connection to the database.  
  
**For Example:**

//Creating connection string to sample database.  
cn.ConnectionString = "Data source=.; Initial Catalog=Sample; User Id=sa; Password=faculty";

The connection string provides the information that defines the connection to the database.

* **Data Source:** Specifies the provider name or your server name.
* **Initial Catalog:** Specifies the name of the database.
* **User Id and Password:** Provide the username and password of your database server.

1. **Open the Connection**

// Creating object of SqlConnection Class.

SqlConnection cn = new SqlConnection();

//Creating connection string to sample database.

cn.ConnectionString = "Data source=.; Initial Catalog=Sample; User Id=sa; Password=faculty";

cn.Open();  // it open the connection to database server..  
   
**b. Close the Connection**

// Creating object of SqlConnection Class.

SqlConnection cn = new SqlConnection();

//Creating connection string to sample database.

cn.ConnectionString = "Data source=.; Initial Catalog=Sample; User Id=sa; Password=faculty";

cn.Open();  // it open the connection to database server..

//Creating sqlcommand class object

SqlCommand cmd = new SqlCommand("Select \* from tblEmployees", cn);

SqlDataReader dr = cmd.ExecuteReader();//Executing query

cn.Close();//Closing the connection

**4.Reading the Data**

A SqlDataReader is a type that is good for reading data in the most efficient manner possible. You can \*not\* use it for writing data. SqlDataReaders are often described as fast-forward firehose-like streams of data.

You can read from SqlDataReader objects in a forward-only sequential manner. Once you’ve read some data, you must save it because you will not be able to go back and read it again.

The forward only design of the SqlDataReader is what enables it to be fast. It doesn’t have overhead associated with traversing the data or writing it back to the data source. Therefore, if your only requirement for a group of data is for reading one time and you want the fastest method possible, the SqlDataReader is the best choice. Also, if the amount of data you need to read is larger than what you would prefer to hold in memory beyond a single call, then the streaming behavior of the SqlDataReader would be a good choice.

using System;

using System.Data;

using System.Data.SqlClient;

public partial class Readform : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

//Step 1

SqlConnection con = new SqlConnection("Data Source=.\\SQLEXPRESS;AttachDbFilename="+ Server.MapPath("App\_Data\\datareader.mdf")+";Integrated Security=True;User Instance=True");

//Step 2

SqlCommand cmd = new SqlCommand("Select \* from Table1", con);

//Step 3

con.Open();

//Step 4

SqlDataReader dr = cmd.ExecuteReader();

// cmd.ExecuteNonQuery(); Insert, Update, Delete

//cmd.ExecuteScalar();

GridView1.DataSource = dr;

GridView1.DataBind();

//Step 5

con.Close();

}

**5. Preparing and Executing the command**

The Command Object uses the connection object to execute SQL queries.

1. The queries can be in the Form of Inline text, Stored Procedures or direct Table access.
2. An important feature of Command object is that it can be used to execute queries and Stored Procedures with Parameters.
3. If a select query is issued, the result set it returns is usually stored in either a DataSet or a DataReader object.

**Properties of SqlCommand class:** The properties associated with SqlCommand class are shown below:

|  |  |  |
| --- | --- | --- |
| **Property** | **Type of Access** | **Description** |
| **Connection** | **Read/Write** | The SqlConnection object that is used by the command object to execute SQL queries or Stored Procedure. |
| **CommandText** | **Read/Write** | Represents the T-SQL Statement or the name of the Stored Procedure. |
| **CommandType** | **Read/Write** | This property indicates how the CommandText property should be interpreted.  1.   1. Text (T-SQL Statement)  2. StoredProcedure (Stored Procedure Name) |
| **CommandTimeout** | **Read/Write** | This property indicates the time to wait when executing a particular command.  **Default Time for Execution of Command is 30 Seconds.** |

**Execute Methods** that can be called from a Command Object.

|  |  |
| --- | --- |
| **Property** | **Description** |
| **ExecuteNonQuery** | This method executes the command specifies and returns the number of rows affected. |
| **ExecuteReader** | The ExecuteReader method executes the command specified and returns an instance of instance of SqlDataReader class. |
| **ExecuteScalar** | This method executes the command specified and returns the first column of first row of the result set. The remaining rows and column are ignored. |
| **ExecuteXMLReader** | This method executes the command specified and returns an instance of XmlReader class. This method can be used to return the result set in the form of an XML document |

**6.Using Parameters with Queries**

Using a parameter in a query is as easy as creating a query that uses criteria. You can design a query to prompt you for one piece of information, such as a part number, or for more than one piece of information, such as two dates. For each parameter, a parameter query displays a separate dialog box that prompts you for a value for that parameter.

**Add a parameter to a query**

1. Create a select query, and then open the query in Design view.
2. In the **Criteria** row of a field for which you want a parameter applied, type the text that you want the parameter dialog box to display, enclosed in square brackets, for example:

When you run the parameter query, the prompt appears in a dialog box without the square brackets.

You can also use an expression with your parameter prompts, for example:

1. Repeat step 2 for each field that you want to add parameters to.

You can use the preceding steps to add a parameter to any one of the following types of queries: Select, Crosstab, Append, Make-table, or Update.

You can also add a parameter to a union query by following these steps:

1. Open the union query in SQL view.
2. Add a WHERE clause that contains each of the fields for which you want to prompt for a parameter.

If a WHERE clause already exists, check to see whether the fields for which you want to use a parameter prompt are already in the clause, and if not, add them.

1. Instead of using a criterion in the WHERE clause, use a parameter prompt.

using System.Data;

using System.Data.SqlClient;

public partial class RFDform : System.Web.UI.Page

{

SqlConnection con;

SqlCommand com;

protected void Page\_Load(object sender, EventArgs e)

{

con = new SqlConnection("Data Source=.\\SQLEXPRESS;AttachDbFilename="+Server.MapPath("App\_Data\\Database.mdf")+";Integrated Security=True;User Instance=True");

con.Open();

}

protected void Button2\_Click(object sender, EventArgs e)

{

TextBox1.Text="";

TextBox2.Text="";

TextBox3.Text="";

TextBox4.Text="";

}

protected void Button1\_Click(object sender, EventArgs e)

{

try

{

com = new SqlCommand("insert into Table1 values(@a,@b,@c,@d)", con);

com.Parameters.AddWithValue("@a", TextBox1.Text);

com.Parameters.AddWithValue("@b", TextBox2.Text);

com.Parameters.AddWithValue("@c", TextBox3.Text);

com.Parameters.AddWithValue("@d", TextBox4.Text);

com.ExecuteNonQuery();

Response.Redirect("RFDform.aspx");

}

catch (Exception ex)

{

Label22.Text = ex.Message;

}

}

}